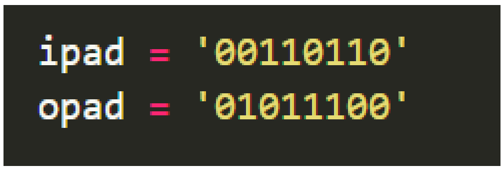
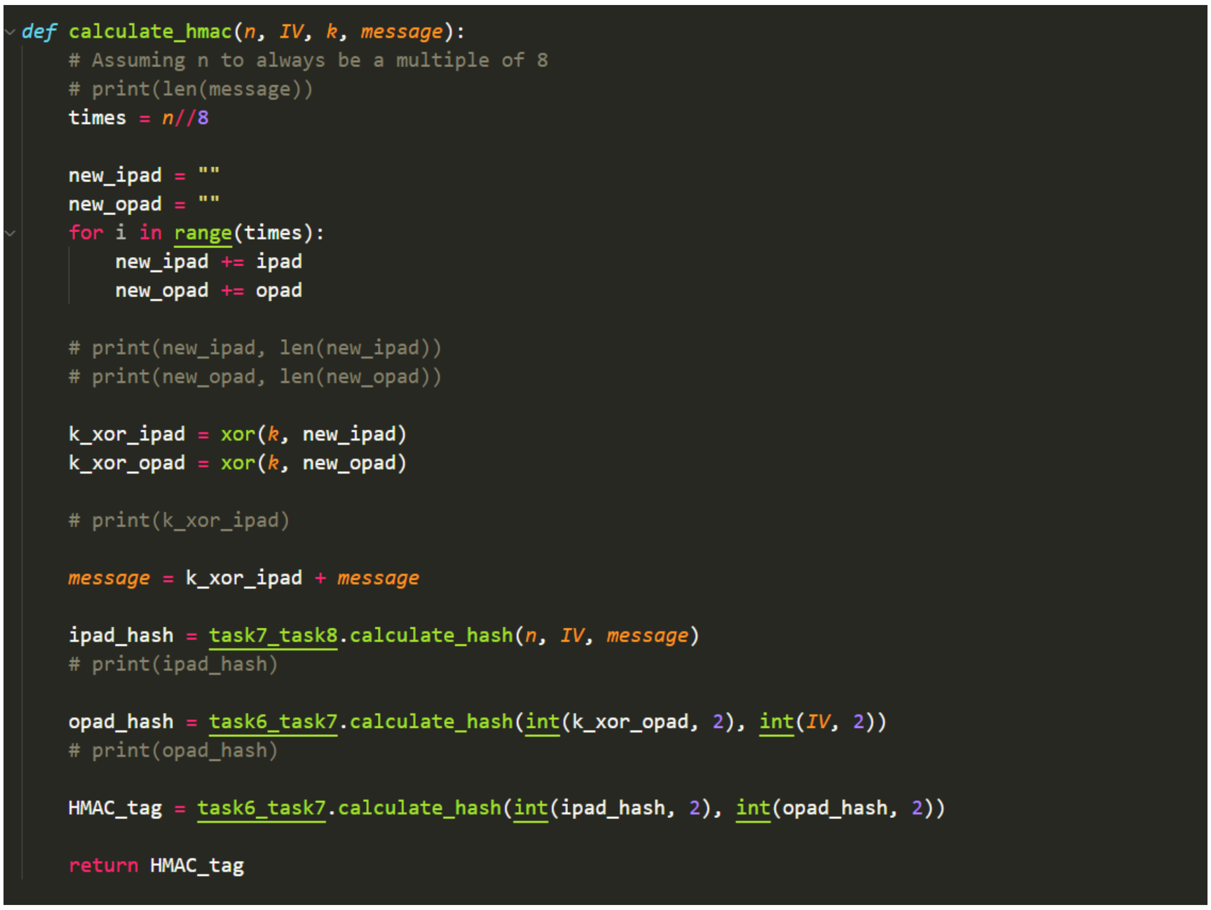
Task 8 HMAC



```python
ipad = '00110110'
opad = '01011100'
```

Ipad: 0x5C

Opad: 0x36



```python
def calculate_hmac(n, IV, k, message):
    # Assuming n to always be a multiple of 8
    # print(len(message))
    times = n//8

    new_ipad = ""
    new_opad = ""
    for i in range(times):
        new_ipad += ipad
        new_opad += opad

    # print(new_ipad, len(new_ipad))
    # print(new_opad, len(new_opad))

    k_xor_ipad = xor(k, new_ipad)
    k_xor_opad = xor(k, new_opad)

    # print(k_xor_ipad)

    message = k_xor_ipad + message

    ipad_hash = task7_task8.calculate_hash(n, IV, message)
    # print(ipad_hash)

    opad_hash = task6_task7.calculate_hash(int(k_xor_opad, 2), int(IV, 2))
    # print(opad_hash)

    HMAC_tag = task6_task7.calculate_hash(int(ipad_hash, 2), int(opad_hash, 2))

    return HMAC_tag
```

Calculate_hmac()

The function takes the fixed length hash size n. Initialization Vector, Key k, and message as input. It pads ipad and opad to get them to the length required. The function then calls methods created under task 6 and task 7 respectively to get the final result.

Sample Input

Fixed hash length: 16

IV: 1001110000110101

Key: 1101000011110111

Message: 101011111111000000011010111000011010101101

Sample Output

HMAC: 0011100001001110